Classification Over Domain Adaptation: a Standard Way

Giulio Bagnoli

January 2020

Abstract: The paper presents an introduction to Domain Adaptation techniques and later a practical application of DANN. It will be applied on AlexeNet and test on the PACS dataset.

1 Domain Adaptation

Domain adaptation (DA) is a sub-discipline of machine learning which deals with scenarios in which a model trained on a source distribution is used in the context of a different but related, target distribution. In general, domain adaptation uses labeled data in one or more source domains to solve new tasks in a target domain. The level of relatedness between the source and target domains usually determines how successful the adaptation will be.

Let X be the input space and let Y be the output space. Let $D = \{X,\ P(x)\}$ a domain where $x \in X$ and $P(x)$ is the data distribution. The task of a classification algorithm is to learn a mathematical model $h : X \to Y$ able to attach a label from Y to an example from X, which, in mathematical terms, can be describe as $T = \{y,\ P(y|x)\}$.

Usually, the samples on which the model is trained and those on which it is tested are drawn from the same domain, so they are *i.i.d.*, but as it is very expensive to obtain large amounts of data necessary to realize working models, it happens that the two sets of data come from different domains.

Let D^s and D^t the domains from which the samples are drawn, respectively, to train the model and to test it. We have that $D^s \neq D^t$ but $T^s = T^t$, it means that the sample representation is different but the task is the same, so the domains have the same labels and $P(y^s|x^s) = P(y^t|x^t)$.

The goal is then to learn h (from labeled or unlabelled samples coming from the two domains) such that it commits as little error as possible on the target domain D^t.

The cases in which it is possible to apply domain adaptation could be divided according to:

The kind of source domain:

- **Single Source domain**, the samples are *i.i.d.*
- **Multi Source domain**, the domains are divided by domain label, so each D^s has the same Y but there is a different $P(y|x)$.
- **Mixed Source domain**, there are several data distributions but the samples are not divided by distribution. It is the most difficult case

The type of data available from the target domain:

- **Supervised DA**, in the D^t are present only labeled data, albeit they are an amount which is too small for training a whole model.
- **Semi-supervised DA**, in the D^t are present both labeled and unlabeled data).
- **Unsupervised DA**, in the D^t are present only unlabeled data.

Domain adaptation techniques can be also divided according to the number of the domain that is pass through to go from the source domain to the target domain. The transaction could be occurred in one step (*one-step domain adaptation*), or in multiple steps (*multi-step domain adaptation*).

The situations in which DA could be applied, according to the labels present in the domains, are two:

- **Closed set**, all labels are present in both source and target domain.

- **Open set**, there are two cases of this scenario. In the first, some of the labels present in the source domain are not present in the target domain. While in the second some of the labels present in the target domain are not present in the source domain.

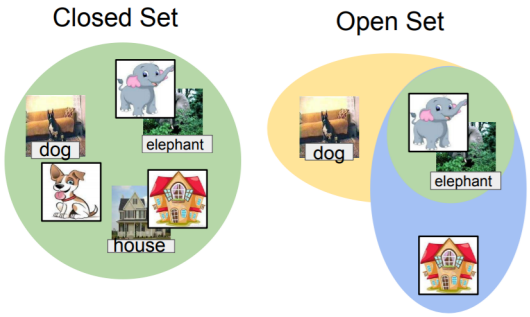

Figure 1: Label overlap.

It is possible to perform domain adaptation techniques in both shallow and deep learning methods. Results obtained in the first case are worse than the second case because shallow learning algorithms classify only the samples whereas deep methods also extract the features, making the model more robust.

Some of the most common techniques to do DA are *Trasfert Component Analysis*, which is used for shallow learning and *Domain-Adversarial Neural Network* (DANN), which is used for a generic architecture. The latter one was used to lead the homework.

2 Domain-Adversarial Neural Network

DANN is a data augmentation technique that allows domain transfer, conducting predictions based on features that cannot discriminate between the source and target domains. This idea is implemented in the context of neural networks that are trained on labeled data from the source domain and unlabeled data from the target domain (Unsupervised DA).

The proposed technique builds mappings between the source and the target domains so that the knowledge learned by the classifier for the source domain can also be applied to the target domain. To do that the adaptation is achieved through aligning the distributions of features across the two domains. However, unlike previous approaches, the alignment is accomplished through standard backpropagation training. In practice, the only non-standard component of the proposed architecture is a rather trivial gradient reversal layer (GRL) defined as follows. The gradient reversal layer has no parameters associated with it. During the forward propagation, the GRL acts as an identity transformation. During the backpropagation, however, the GRL takes the gradient from the subsequent level and changes its sign, *i.e.*, multiplies it by -1, before passing it to the preceding layer.

So during the training process, the model learns to extract features that are:

- Discriminatory for the main learning task on the source domain.
- Indiscriminate respect to the shift between the domains.

To learn this kind of features the model has two different classifiers:

- The *label predictor* that predicts class labels. It is used both during training and at test time.
- The *domain classifier* that discriminates between the source and the target domains during training.

which are jointly optimize.

So the parameters of the network are optimized to minimize the loss of the label predictor and to maximize the loss of the domain classifier. The latter update thus works adversarially to the domain classifier, and it encourages domain-invariant features to emerge in the course of the optimization.

A network composed in this way is called: *domain-adversarial neural network*.

The idea of maximizing the loss of the domain classifier and of minimizing the loss of the label predictor comes from this article[1] in which the authors explains that, to train a model that can generalize well from one domain to another, the internal representation of the neural network has to contain no discriminative information about the origin of the input (source or target), while preserving a low risk on the source (labeled)

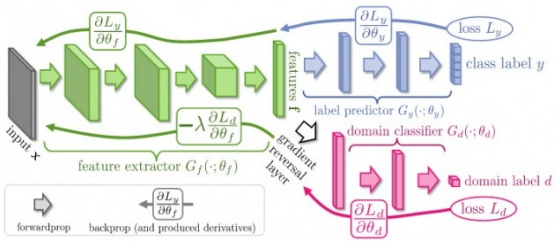

Figure 2: Domain-adversarial neural network. The proposed architecture includes a deep feature extractor (green) and a deep label predictor (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a domain classifier (red).

examples.

From a mathematical point of view, let $G_f(\cdot\ ,\ \theta_f)$ be the D-dimensional neural network feature extractor, with parameters θ_f. Also, let $G_y(\cdot\ ,\ \theta_y)$ be the part of DANN that computes the network's label prediction output layer, with parameters θ_y, while $G_d(\cdot\ ,\ \theta_d)$ corresponds to the computation of the domain prediction output of the network, with parameters θ_d.

The prediction loss and the domain loss are respectively defined by:

$$L_y^i(\theta_f,\ \theta_y) \ = \ L_y(G_y(G_f(x_i,\ \theta_f),\ \theta_y),\ y_i) \quad (1)$$

$$L_d^i(\theta_f,\ \theta_d) \ = \ L_d(G_d(G_f(x_i,\ \theta_f),\ \theta_d),\ y_i) \quad (2)$$

So training DANN consists of minimizing:

$$E(\theta_f,\theta_y,\theta_d) = \sum_{\substack{i=1\ldots N\\ d_i=0}} L_y(G_y(G_f(x_i,\theta_f),\theta_y),y_i) - \lambda \sum_{i=1\ldots N} L_d(G_d(G_f(x_i,\theta_f),\theta_d),y_i)$$

$$= \sum_{\substack{i=1\ldots N\\ d_i=0}} L_y^i(\theta_f,\ \theta_y) - \lambda \sum_{i=1\ldots N} L_d^i(\theta_f,\ \theta_d) \quad (3)$$

by finding the saddle point $(\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d)$ such that:

$$(\hat{\theta}_f, \hat{\theta}_y) = arg \min_{\theta_f,\theta_y} E\Big(\theta_f, \theta_y, \hat{\theta}_d\Big) \quad (4)$$

$$\hat{\theta}_d = arg \max_{\theta_d} E\Big(\hat{\theta}_f, \hat{\theta}_y, \theta_d\Big) \quad (5)$$

As shown in [2] saddle point defined by Equations (4) (5) can be found as a stationary point of the following gradient updates:

$$\theta_f \leftarrow \theta_f - \eta\Big(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}\Big) \quad (6)$$

$$\theta_y \leftarrow \theta_y - \eta \frac{\partial L_y^i}{\partial \theta_y} \quad (7)$$

$$\theta_d \leftarrow \theta_d - \eta\lambda \frac{\partial L_d^i}{\partial \theta_d} \quad (8)$$

where η is the learning rate. The updates of Equations (6-8) are very similar to stochastic gradient descent (SGD) updates for a feed-forward deep model that comprises a feature extractor fed into the label predictor and into the domain classifier (with loss weighted by λ). The only difference is that in (6), the gradients from the class and domain predictors are subtracted, instead of being summed. This is due to the fact that it is necessary to maximize the loss on the domain classifier. This difference in SGD calculation is implemented by introducing the *gradient reversal layer*. For this reason, GRL is inserted between the features extractor G_f and the domain classifier G_d building, in this way, the architecture in figure 2.

Therefore, running SGD in the resulting model implements the updates of Equations (6-8) and converges to a saddle point of Equation (3).

3 Description of the project

3.1 Code Overview

The purpose of the experiment is to show how the application of DANN to AlexNet improves the network performance in the presence of domain shift. To do this AlexNet's implementation on Pythorch has been modified, the new class is called DANN_AlexNet. The changes made to the old class are:

- The addition of the domain classifier, which is equal to the AlexNet label predictor, with only two neurons as output, as it must distinguish between two domains.
- The insertion of GRL between the domain classifier and the convolutional part of the network, so that the backpropagation works properly.
- The modification of the forward step so that the network can pass the extracted features to the correct classifier.
- As AlexNet's pretrained model will be used during the experiment, the override of the method that handles the loading of AlexNet pretrained has been implemented. The new method loads the weights of the label predictor of AlexNet pretrained also on the domain classifier.

The experiment was conducted on the data contained in the PACS dataset, which contains 4 different domains: Art painting, Cartoon, Photo and Sketch. The class *PACS* and the method *prepareDataloader*, that makes 4 dataloader, one for each domain, were developed to work with this dataset. Other notable code parts are:

- *test_accuracy_NO_DANN*, the function trains the network for 30 epochs. During the experiment, the photo domain is used as the source domain. At each epoch, the accuracy of the model on the Cartoon and Sketch domains is tested and averaged. Once finished *test_accuracy_NO_DANN* returns: the model that has obtained the highest average accuracy and its hyperparameters (*eta*, *step_size*, number of epochs for which the model has been trained). Since the method does not involve the use of DANN the parameters of the domain classifier are not passed to the optimizer.
- *test_accuracy_DANN*, it performs a task similar to the function described above, with the only difference that DANN is applied. Three steps are conducted to do this during the training phase as is shown in Alg. 1. In the first step, the label predictor is trained by passing it to the source domain samples. In the second one, the domain classifier is trained by passing the data of the previous step having the label equals zero. In the third one, the domain classifier is trained by passing it the data of the "validation" domain having the label equals to one.

  Contrary to the previous method the evolution of the network during the epochs depends not only on the data present in the source domain but also on those present in the "validation" domain, for this reason, accuracy at each time is calculated only on data belonging to a single domain. The method returns a list containing these accuracies.
- *LOG*, the class used to print the details on the trend of the losses during the execution for point 4. The method *print_log* reports the various losses during the phases of the execution while the method *print_loss* prints the losses in an easily usable format to print graphics on Python.

  In this paper, the charts of the losses have not been brought back but they were used during the development of the homework to verify the trend of the losses. In general, for the training of 30 epochs, the loss on the classifier of domain tends to be lowered in the first epochs and then to go up slightly until to stabilize on values of the order of $10^0 - 10^1$.

for *i=0; To NUM_EPOCH* **do**
  for *batch in source_dataloader* **do**
    1) Train on source labels by forwarding source data to G_y + loss.backward()
    2) Train the domain discriminator by forwarding source data to G_d + loss.backward()
    3) Train the domain discriminator by forwarding validation data to G_d + loss.backward()
  end
  Scheduler.step()
  Accuracy calculation on "validation" domain.
end

Algorithm 1: Pseudocode of *test_accuracy_DANN*.

3.2 The Experiment

Experiments are conducted to evaluate the effectiveness of DANN on the PACS dataset, in particular the PACS domains have been used as follows:

- Photo → Source
- Art Painting → Target
- Cartoon → "Validation"
- Sketch → "Validation"

Initially, the performances of the trained model are compared on the Photo domain and evaluated on the Art Painting domain with both DANN and without. The combination of hyperparameters used is ($\eta = 0.0001, step\_size = 25, num\_epoch = 30, \lambda = 1$), for these values the application of DANN improves, going from an accuracy of 45.3% to 48.17%. The data reported during this phase were obtained from an average of 20 iterations of the experiment.

The fact that the use of DANN does not significantly improve the data may be due both to the fact that:

- λ is equal to 1 for the whole training process and this allows the domain classifier to be more sensitive to noisy signals at the early epochs.
- The hyperparameters combination, that was chosen a priori without evaluating the model on a validation set, is not the best.

To avoid the first problem the authors of [2] suggests to use two different factor ($\lambda_f c$, λ) depending on which network component is updated. In the article $\lambda_f c$ is initiated at 0 its values reaches 1 using the following schedule:

$$\lambda_f c = \frac{2}{1 + exp(-\gamma \cdot p)} - 1 \tag{9}$$

where p is the training progress linearly changing from 0 to 1 and γ is a fixed value.

$\lambda_f c$ is the factor that is used for updating the feature extractor component G_f. While $\lambda = 1$ is used for updating the domain classification component, to ensure that the latter trains as fast as the label predictor G_y.

To avoid the second problem it is possible to do a tune of hyperparameters, considering Cartoon and Sketch domains as validation sets. A grid search for hyperparameters (η, $step\_size$) has been performed both in case DANN is applied and without. In the first case also λ was tuned.

The hyperparameters values used during the experiment without domain adaptation are:

- $\eta \in [0.01,\ 0.005,\ 0.001]$
- $step\_size \in [25,\ 30]$

In the case of domain adaptation the hyperparameters values used are a bit different because it perform better with lower η, the values used are:

- $\eta \in [0.001,\ 0.0005,\ 0.0001]$
- $step\_size \in [25,\ 30]$
- $\lambda \in [0.5,\ 0.3,\ 0.2]$

Experiments for λ values greater than 0.5 have not been reported as the loss on the label predictor, after a few epochs, assumed the value *"NaN"*.

To choose the best combination of hyperparameters, the average accuracy calculated on the two validation sets is used as evaluation criteria. The following data are the average of data obtained from three different executions.

The best averaged accuracy obtained calculated on the two "validation" sets without domain adaptation, fluctuates between 24.48% and 28.87%, data are shown in the Table 1. The highest values are obtained with the highest η.

The low results obtained on the Cartoon and Sketch domains are caused by the difference between the distributions of the data in such domains and the data of the source domain. While the target accuracy is around 43.79%.

η	$step\_size$	Accuracy
0.001	25	28.12%
0.001	30	28.87%
0.0005	25	27.86%
0.0005	30	26.6%
0.0001	25	24.75%
0.0001	30	24.48%

Table 1: The best and the worst averaged values.

η	*step_size*	λ	Accuracy
0.001	25	0.5	39.50%
0.001	30	0.1	26.5%
0.0001	25	0.5	23.63%
0.0001	30	0.1	26.5%

Table 2: Some averaged accuracy on "validation" domains.

The result obtained by hyperparameters validation is worse than the previous one obtained without it, this is probably due to the fact that the data distribution of the validation and the target domain are different and so the set of hyperparameters that is the best in a domain could not be the best in another one.

With the use of DANN, the results improve slightly, the accuracy on the target domain reaches values around 50%.

The increase in accuracy is not extremely high, only by 5%. The causes of this could be:

- the samples of the source domain are few, only 1600.
- the choice of the best combination of the hyperparameters depends on a different distribution of data than that of the target domain.

To solve the first problem, data augmentation techniques could be implemented.

References

[1] Shai Ben-David, John Blitzer, and Koby Crammer. *A theory of learning from different domains.* URL: `https://www.alexkulesza.com/pubs/adapt_mlj10.pdf`.

[2] Yaroslav Ganin et al. *Domain-Adversarial Training of Neural Networks.* URL: `https://arxiv.org/pdf/1505.07818.pdf`.